GALAHAD LANCELOT_simple

USER DOCUMENTATION GALAHAD Optimization Library version 5.1

1 SUMMARY

LANCELOT_simple is a Fortran module for minimizing an objective function, where the minimization variables are required to satisfy a set of auxiliary, possibly nonlinear, constraints. Bounds on the variables and known values may be specified. The module povides access to a single subroutine, called LANCELOT_simple, which in turn provides a very simple calling sequence for the much more powerful LANCELOT B pcakage. Its simplicity results from the fact that LANCELOT_simple completely ignores partial separability and sparsity structure, limits the forms under which the problem can be presented to the solver and only allows for a very restricted choice of algorithmic parameters. It is therefore of interest mostly for small-dimensional problems for which ease of interface matters more than numerical performance.

1.1 A brief introduction to terminology and scope

We consider the nonlinear minimization problem given by

$$\min_{x \in \mathbb{R}^n} f(x),$$

possibly subject to constraints of the one or more of the forms

$$l \leq \mathbf{x} \leq u,$$

$$c_e(\mathbf{x}) = 0,$$

$$c_i(\mathbf{x}) \leq 0,$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $c_e : \mathbb{R}^n \to \mathbb{R}^m$ and $c_i : \mathbb{R}^n \to \mathbb{R}^q$ are twice-continuously differentiable functions, and l, u are vectors of $\mathbb{R}^n$ whose components are allowed to be arbitrarily large in absolute value.

The inequality constraints are first internally reformulated as equalities by the introduction of (non-negative) slack variables. This defines the set $\mathcal{C}$ of all (original and reformulated) equality constraints. The method used to solve this reformulated problem is iterative and features two levels of iteration. In the outer level, a composite function, the augmented Lagrangian merit function,

$$\phi(\mathbf{x}, \mathbf{y}, \mu) = f(\mathbf{x}) + \sum_{i \in \mathcal{C}} y_i c_i(\mathbf{x}) + \frac{1}{2\mu} \sum_{i \in \mathcal{C}} [c_i(\mathbf{x})]^2, \tag{1.1}$$

is formulated, where μ is known as the penalty parameter, $\mathbf{y}$ is a vector of Lagrange multiplier estimates and c_i now ranges over all (original and reformulated) equality constraints. Each outer iteration requires the approximate minimization of this merit function within the feasible box, for given values of μ and $\mathbf{y}$.

The required approximate minimization for fixed μ and $\mathbf{y}$ is carried out using a series of inner iterations. At each inner iteration, a quadratic model of the merit function is constructed. An approximation to the minimizer of this model within a trust-region is calculated. The trust region is a "box" of specified radius, centered at the current best estimate of the minimizer. If there is an accurate agreement between the model and the true objective function at the new approximation to the minimizer, this approximation becomes the new best estimate. Otherwise, the radius of the trust region is reduced and a new approximate minimizer sought. The algorithm also allows the trust-region radius to increase when necessary. The minimization of the model function is carried out by using an iterative approach.

The approximate minimization of the model is performed in two stages. In the first, a so-called generalized Cauchy point is determined by approximately minimizing the model within the intersection of the feasible box and the trust-region along a scaled steepest descent direction. Having taken this step, the model is further reduced by solving one or more quadratic minimization problems in which any constraints activated at the Cauchy point remain so. The latter computation is essentially equivalent to the solution of a sequence of linear systems, and is performed using an iterative (preconditioned conjugate gradient) method.

After an appropriate approximation to the minimizer of the merit function is obtained, and if there are general constraints, μ and $\mathbf{y}$ are adjusted to ensure convergence of the outer iteration to the required solution of the constrained minimization problem.

ATTRIBUTES — Versions: `LANCELOT_simple_double`, **Uses:** `LANCELOT`. **Date:** November 2007. **Origin:** N. I. M. Gould, Oxford University and Rutherford Appleton Laboratory, England, D. Orban, Ecole Polytechnique, Montréal, Canada, and Ph. L. Toint, University of Namur - FUNDP, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE LANCELOT_simple_single
```

with the obvious substitution `LANCELOT_simple_double`, `LANCELOT_simple_quadruple`, `LANCELOT_simple_single_64`, `LANCELOT_simple_double_64` and `LANCELOT_simple_quadruple_64` for the other variants.
If it is required to use more than one of the modules at the same time, the subroutine `LANCELOT_simple` (Section 2.3) must be renamed on one of the `USE` statements.

2.2 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.3 Argument lists and calling sequences

The minimization subroutine `LANCELOT_simple` is called as follows:

```
CALL LANCELOT_simple( n, X, MY_FUN, fx, exit_code [,MY_GRAD] [,MY_HESS]    &
                      [,BL] [,BU] [,VNAMES] [,CNAMES]                      &
                      [,neq] [,nin] [,CX] [,Y] [,iters] [,maxit]           &
                      [,gradtol] [,feastol] [,print_level] )
```

where

`n` is a scalar variable of type `INTEGER(ip_)`, that holds the number of optimization variables, n which must all be set on entry. It will thereafter be unaltered.

X is a rank-one array of dimension n and type REAL(rp_), that holds the current values of the minimization variables, **x**. On input, it must contain the values of the variables corresponding to the minimization starting point. On output, it contains the best point found by the routine.

MY_FUN is a variable whose value is the name of a user-supplied subroutine whose purpose is to compute objective function and constraint values. See Section 2.4.1 for details. The subroutine associated with the variable must be declared EXTERNAL in the calling program.

fx is a scalar variable of type REAL(rp_), that holds on output the value of the objective function f at X.

exit_code is a scalar variable of type INTEGER(ip_), that holds on output the final status for the minimization, a value 0 indicating success. Other values are described below (see Section 2.5).

MY_GRAD is an OPTIONAL variable whose value is the name of a user-supplied subroutine whose purpose is to compute the first derivatives of the objective function and constraints. See Section 2.4.2 for details. If MY_GRAD is present the subroutine associated with the variable must be declared EXTERNAL in the calling program. If the argument is not present, first derivatives will be estimated by finite differences.

MY_HESS is an OPTIONAL variable whose value is the name of a user-supplied subroutine whose purpose is to compute the second derivatives of the objective function and constraints. See Section 2.4.3 for details. If MY_HESS is present the subroutine associated with the variable must be declared EXTERNAL in the calling program. If the argument is not present, or if MY_GRAD is not present, second derivatives will be estimated by secant formulae.

BL is an OPTIONAL rank-one array of dimension n and type REAL(rp_), whose i-th entry may be set to the value of the lower bound l_i on the i-th variable. If the i-th variable has no lower bound, BL(i) should be set to a large negative number. It is not altered by the routine.

BU is an OPTIONAL rank-one array of dimension n and type REAL(rp_), whose i-th entry may be set to the value of the upper bound u_i on the i-th variable. If the i-th variable has no upper bound, BU(i) should be set to a large positive number. It is not altered by the routine.

CNAMES is an OPTIONAL rank-one array of dimension neq + nin and type default CHARACTER and length 10, whose i-th entry contains (on input) the "name" of the i-th equality constraint for i = 1, ..., neq and of the ith inequality constraint for i = neq+1, ..., neq+nin. It is not altered by the routine.

VNAMES is an OPTIONAL rank-one array of dimension n and type default CHARACTER and length 10, whose j-th entry contains (on input) the "name" of the j-th variable. It is not altered by the routine.

neq is an OPTIONAL scalar variable of type INTEGER(ip_), that holds the number of equality constraints. If not present on input, it is assumed that there are no inequality constraints neq = 0). It is not altered by the routine.

nin is an OPTIONAL scalar variable of type INTEGER(ip_), that holds the number of inequality constraints. If not present on input, it is assumed that there are no inequality constraints nin = 0). It is not altered by the routine.

CX is an OPTIONAL rank-one array of dimension neq + nin and type REAL(rp_), whose i-th component holds on output the current estimates of the values of the equality constraints (i = 1, ..., neq), and of the inequality constraints (i = neq+1, ..., neq+nin).

Y is an OPTIONAL rank-one array of dimension neq+nin and type REAL(rp_), whose i-th component holds on output the current estimates of the Lagrange multipliers, **y**, for the equality constraints (i = 1, ..., neq), and for the inequality constraints (i = neq+1, ..., neq+nin).

iters is an OPTIONAL scalar variable of type INTEGER(ip_), that gives on output the number of iterations that have been performed since the start of the minimization.

maxit is an OPTIONAL scalar variable of type INTEGER(ip_), that holds the maximum number of iterations which will be allowed in the solver. The default is maxit = 1000.

gradtol is an OPTIONAL scalar variable of type REAL(rp_), that is used to specify on input the maximum permitted (infinity) norm of the projected gradient of the Lagrangian function (see Section 4) at the estimate of the solution sought. The default is gradtol = 10^{-5}. It is not altered by the routine.

feastol is an OPTIONAL scalar variable of type REAL(rp_), that is used to specify on input the maximum permitted violation (measured in infinity norm) of the constraints at the estimate of the solution sought. The default is feastol = 10^{-5}.

print_level is an OPTIONAL scalar variable of type INTEGER(ip_), that is used to control on input the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level $= 1$, a single line of output will be produced for each iteration of the process, while additionally if print_level $= 2$ a summary of the inner iteration will be given. If print_level ≥ 3, this output will be increased to provide significant detail of each iteration. The default is print_level = 1.

2.4 Function and derivative values

2.4.1 Evaluating problem functions

LANCELOT_simple requires that the user provides a subroutine, with prescribed argument lists, that accept input values ($\mathbf{x}$), and provide as output $f(\mathbf{x})$) or $c_i(\mathbf{x})$ ($i \in \mathcal{C}$). This default routine name is FUN and it must have the following argument list:

```
SUBROUTINE FUN ( X, fx [,i] )
```

where

X is a rank-one INTENT(IN) array argument of dimension n and type REAL(rp_), that contains the values of $\mathbf{x}$ at which the subroutine is required to evaluate the values of the objective or constraint functions.

fx is a rank-one INTENT(OUT) scalar argument of type REAL(rp_), that contains the value of the relevant function evaluated at $\mathbf{x}$.

i is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip_), that, if present, specifies the index of the constraint function to be evaluated (i = 1, ..., neq+nin). The values of i between 1 and neq correspond to equality constraints, those between neq+1 and neq+nin to inequality constraints. If i is not present, this indicates that the value of the objective function must be evaluated.

The name of the function evaluation routine may be modified by the user by specifying MY_FUN to a value different from FUN in the calling sequence, but the argument list of the associated subroutine must be identical to that described above. Specifying MY_FUN = FUN is redundant in the calling sequence if the name of the function evaluation routine is FUN (but this explicit identification is nevertheless recommended for clarity).

2.4.2 Evaluating the first derivatives of problem functions

If the first derivatives of the problem functions are available, which is indicated by the specification of the OPTIONAL argument MY_GRAD, they must be available for the objective function and all constraints. LANCELOT_simple then requires that the user provide a subroutine, with a prescribed argument list, that accepts input values ($\mathbf{x}$), and returns as output $\nabla_x f(\mathbf{x})$, $\nabla_x c_e(\mathbf{x})$ or $\nabla_x c_i(\mathbf{x})$. This routine is specified by setting the value of the OPTIONAL argument MY_GRAD. If one uses MY_GRAD = GRAD in the call to LANCELOT_simple, the GRAD routine must have the following argument list:

```
SUBROUTINE GRAD ( X, G [,i] )
```

where

X is a rank-one INTENT(IN) array argument of dimension n and type REAL(rp_), that contains the values of **x** at which the subroutine is required to evaluate the gradient of the objective or constraint function.

G is a rank-one INTENT(OUT) array argument of dimension n and type REAL(rp_), that contains the value of the relevant gradient evaluated at **x**.

i is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip_), that, if present, specifies the index of the constraint function whose gradient must be evaluated (i = 1, ..., neq+nin). The values of i between 1 and neq correspond to equality constraints, those between neq+1 and neq+nin to inequality constraints. If i is not present, this indicates that the gradient of the objective function must be evaluated.

2.4.3 Evaluating the second derivatives of problem functions

If the second derivatives of the problem functions are available, which is indicated by the specification of the OPTIONAL argument HESS, they must be available for the objective function and all constraints. Moreover, first derivatives must also be available for all problem functions, and must be specified as indicated in Section 2.4.2. LANCELOT_simple then requires that the user provide a subroutine, with a prescribed argument list, that accepts input values $(\mathbf{x})$, and returns as output $\nabla_{xx} f(\mathbf{x})$, $\nabla_{xx} c_e(\mathbf{x})$ or $\nabla_{xx} c_i(\mathbf{x})$. This routine is specified by setting the value of the OPTIONAL argument MY_HESS. If one uses MY_HESS = HESS in the call to LANCELOT_simple, the HESS routine must have the following argument list:

```
SUBROUTINE HESS ( X, H [,i] )
```

where

X is a rank-one INTENT(IN) array argument of dimension n and type REAL(rp_), that contains the value of **x** at which the subroutine is required to evaluate the Hessian of the objective or constraint function.

H is a rank-one INTENT(OUT) array argument of dimension n*(n+1)/2 and type REAL(rp_), that contains the value of the relevant Hessian evaluated at **x**. Only the "upper triangular" part of the required Hessians should be specified columnwise. In other words, the component of the Hessian with respect to variables p and j, with $p \leq j$, must be placed in H(j*(j-1)/2 + p).

i is an OPTIONAL scalar INTENT(IN) argument of type INTEGER(ip_), that, if present, specifies the index of the constraint function whose Hessian must be evaluated (i = 1, ..., neq+nin). The values of i between 1 and neq correspond to equality constraints, those between neq+1 and neq+nin to inequality constraints. If i is not present, this indicates that the Hessian of the objective function must be evaluated.

2.5 Warning and error messages

If exit_code is positive on return from the solver, an error has been detected. The user should correct the error and restart the minimization. Possible values of exit_code and their consequences are:

exit_code $= 1$. More than maxit iterations have been performed. This is often a symptom of incorrectly programmed derivatives or of the preconditioner used being insufficiently effective. Recheck the derivatives. Otherwise, increase maxit and re-enter lanb at the best point found so far.

exit_code $= 2$. The trust-region radius has become too small. This is often a symptom of incorrectly programmed derivatives or of requesting more accuracy in the projected gradient than is reasonable on the user's machine. If the projected gradient is small, the minimization has probably succeeded. Otherwise, recheck the derivatives.

exit_code $= 3$. The step taken during the current iteration is so small that no difference will be observed in the function values. This sometimes occurs when too much accuracy is required of the final gradient. If the projected gradient is small, the minimization has probably succeeded.

exit_code = 8. The problem does not appear to have a feasible solution. Check the constraints and try starting with a different initial value for **x**.

exit_code = 15. The problem dimension n is nonpositive. Ensure that n > 0.

exit_code = 19. One or both of the numbers of constraints neq or nin is negative. Ensure that both neq ≥ 0 and nin ≥ 0.

2.6 Information printed

The user is able to control the amount of intermediate printing performed in the course of the minimization. Printing is under the control of the parameter print_level and output is sent to I/O unit number number 6. Possible values of print_level and the levels of output produced are as follows.

print_level ≤ 0. No printing, except warning messages, will be performed.

print_level ≥ 1. Details of the minimization function will be output. This includes the number of variables, and additional information on the internal data structures of the solver.

If the current iterate provides an acceptable estimate of the minimizer of the augmented Lagrangian function, the two-norm of the general constraints and the current value of the penalty parameter are given.

print_level = 1. A simple one-line description of each iteration is given. This includes the iteration number, the number of derivative evaluations that have been made, the number of conjugate-gradient iterations that have been performed, the current value of the augmented Lagrangian function, the (two-) norm of the projected gradient, the ratio ρ of actual to predicted decrease in the augmented Lagrangian function value, the current trust-region radius, the norm of the step taken, an indication of how the direct or iterative method ended, the number of variables which lie away from their bounds and the total time spent on the minimization.

print_level = 2. In addition to the information output with print_level = 1, a short description of the approximate solution to the inner-iteration linear system is given. Before a successful (exit_code = 0) exit, details of the estimate of the minimizer and the gradient of the augmented Lagrangian function are given.

print_level = 3. A list of the current iteration number, the value of the augmented Lagrangian function, the number of derivative evaluations that have been made, the (two-) norm of the projected gradient, the number of conjugate gradients iterations that have been performed and the current trust-region radius are given, followed by the current estimate of the minimizer. The values of the reduction in the model of the augmented Lagrangian function and the actual reduction in this function, together with their ratio, are also given. Before a successful (exit_code = 0) exit, details of the estimate of the minimizer and the gradient of the augmented Lagrangian function are given.

If the current iterate also provides an acceptable estimate of the minimizer of the augmented Lagrangian function, values of the general constraints and estimates of the Lagrange multipliers are also given.

print_level = 4. In addition to the information output with print_level= 3, the gradient of the augmented Lagrangian function at the current estimate of the minimizer is given. Full details of the approximate solution to the inner-iteration linear system are also given. This level of output is intended as a debugging aid for the expert only.

print_level = 5. In addition to the information output with print_level = 4, the diagonal elements of the second derivative approximation are given.

print_level ≥ 6. In addition to the information output with print_level = 5, the second derivative approximations to each problem function are given.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: The user must provide an external `MY_FUN` subroutine (see Section 2.4.1), and optionally may provide external `MY_GRAD` (see Section 2.4.2) and `MY_HESS` (see Section 2.4.3) subroutines.

Other modules used directly: `LANCELOT`.

Input/output: No input; output on device number 6. Output is provided under the control of `print_level`.

Restrictions: `n` $>$ `0`, `neq` $\geq$ `0`, `nin` $\geq$ `0`.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003.

4 METHOD

The basic method implemented within `LANCELOT_simple` and LANCELOT B is described in detail by Conn, Gould and Toint (1991). The method used to solve the inner iteration subproblem is described by Conn, Gould and Toint (1988b). Also see Chapter 3 of the LANCELOT A manual.

The Lagrangian function associated with the objective function and the general constraints is the composite function

$$\ell(\mathbf{x},\mathbf{y}) = f(\mathbf{x}) + \sum_{i \in \mathcal{C}} y_i c_i(\mathbf{x}).$$

The scalars y_i are known as Lagrange multiplier estimates. At a solution $\mathbf{x}^\star$ to the constrained minimization problem, there are Lagrange multipliers $\mathbf{y}^\star$ for which the components of the gradient of the Lagrangian function $\partial \ell(\mathbf{x}^\star,\mathbf{y}^\star)/\partial x_i = 0$ whenever the corresponding variable $x_i^\star$ lies strictly between its lower and upper bounds.

The augmented Lagrangian function is the composite function

$$\phi(\mathbf{x},\mathbf{y},\mu) = \ell(\mathbf{x},\mathbf{y}) + \frac{1}{2\mu}\sum_{i \in \mathcal{C}} [c_i(\mathbf{x})]^2, \tag{4.1}$$

where μ is known as the penalty parameter. An inner iteration is used to find an approximate minimizer of (4.1) within the feasible box for fixed values of the penalty parameter and Lagrange multiplier estimates. The outer iteration of `LANCELOT_simple` automatically adjusts the penalty parameter and Lagrange multiplier estimates to ensure convergence of these approximate minimizers to a solution of the constrained optimization problem.

In the inner iteration, a step from the current estimate of the solution is determined using a trust-region approach. That is, a quadratic model of the augmented Lagrangian function is approximately minimized within the intersection of the constraint "box" and another convex region, the trust-region. This minimization is carried out in two stages. Firstly, the so-called generalized Cauchy point for the quadratic subproblem is found. (The purpose of this point is to ensure that the algorithm converges and that the set of bounds which are satisfied as equations at the solution is rapidly identified.) Thereafter an improvement to the quadratic model is sought using either a direct-matrix or truncated conjugate-gradient algorithm. The trust-region size is increased if the reduction obtained in the objective function is reasonable when compared with the reduction predicted by the model and reduced otherwise.

The strategy for treating bound constraints is based on the usual projection and is described in detail in Conn, Gould and Toint (1988a).

References:

The basic method is described in detail in

A. R. Conn, N. I. M. Gould and Ph. L. Toint (1992). LANCELOT. A fortran package for large-scale nonlinear

optimization (release A). Soringer Verlag Series in Computational Mathematics 17, Berlin,
and details of its computational performance may be found in
A. R. Conn, N. I. M. Gould and Ph. L. Toint (1996). Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization Mathematical Programming **73** 73-110.

Convergence properties of the method are described in
A. R. Conn, N. I. M. Gould and Ph. L. Toint (1991). A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds. SIAM Journal on Numerical Analysis **28** 545-572,
and
A. R. Conn, N. I. M. Gould and Ph. L. Toint (1988a). Global convergence of a class of trust region algorithms for optimization with simple bounds. SIAM Journal on Numerical Analysis **25** 433-460,
while details of the inner iteration are provided by
A. R. Conn, N. I. M. Gould and Ph. L. Toint (1988b). Testing a class of methods for solving minimization problems with simple bounds on the variables. Mathematics of Computation **50** 399-430.

Many of the newer issues are discussed by
A. R. Conn, N. I. M. Gould and Ph. L. Toint (2000). Trust Region Methods. SIAM, Philadelphia.
An easy-to-read introduction to `LANCELOT_simple` is provided in
N. I. M. Gould, D. Orban and Ph. L. Toint (2007). LANCELOT_simple, a simple interface to LANCELOT B. Report 07/12, Department of Mathematics, University of Namur-FUNDP, Namur, Belgium.

5 EXAMPLE OF USE

We now consider the small example problem

$$\min_{x_1,x_2} f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1-x_1)^2,$$

subject to the constraints

$$\begin{array}{rcl} 0 & \leq & x_1, \\ x_1 + 3x_2 - 3 & = & 0, \\ x_1^2 + x_2^2 - 4 & \leq & 0. \end{array}$$

A simple Fortran program to use the interface on this problem is given as follows.

```
      PROGRAM RUN_LANCELOT_simple

      USE LANCELOT_simple_double

      IMPLICIT NONE
      INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
      REAL ( KIND = wp ), PARAMETER :: infinity = 10.0_wp ** 20
      INTEGER :: n, neq, nin, iters, maxit, print_level, exit_code
      REAL ( KIND = wp ) :: gradtol, feastol, fx
      CHARACTER ( LEN = 10 ), ALLOCATABLE, DIMENSION(:) :: VNAMES, CNAMES
      REAL ( KIND = wp ),     ALLOCATABLE, DIMENSION(:) :: BL, BU, X, CX, Y
      EXTERNAL :: FUN, GRAD, HESS
!
! THE TEST PROBLEM DIMENSIONS (user defined)
!
       n   = 2  ! number of variables
       neq = 1  ! number of equality constraints, excluding fixed variables
       nin = 1  ! number of inequality (<= 0) constraints, excluding bounds
!
! allocate space for problem defining vectors
!
```

```
      ALLOCATE( X( n  ), BL( n ), BU( n ), CX( neq+nin ), Y( neq+nin ) )
      ALLOCATE( VNAMES( n  ), CNAMES( neq+nin ) )
!
! starting point
!
      X(1) = -1.2_wp                ! starting point (componentwise)
      X(2) =  1.0_wp
!
! bounds on the variables
!
      BL(1) =  0.0_wp               ! lower bounds (componentwise)
      BL(2) = -infinity
      BU(1) =  infinity             ! upper bounds (componentwise)
      BU(2) =  3.0_wp
!
! names
!
      VNAMES(1) = 'x1'              ! variables
      VNAMES(2) = 'x2'
      CNAMES(1) = 'Equality'        ! equality constraints
      CNAMES(2) = 'Inequality'      ! inequality constraints
!
! algorithmic parameters
!
      maxit       = 100
      gradtol     = 0.00001_wp      ! identical to default
      feastol     = 0.00001_wp      ! identical to default
      print_level = 0               ! no output
!
! solve by calling LANCELOT_simple
!
      CALL LANCELOT_simple( n,  X, FUN, fx, exit_code,                  &
                            MY_GRAD = GRAD , MY_HESS = HESS,            &
                            BL = BL, BU = BU, VNAMES   =  VNAMES,       &
                            CNAMES =  CNAMES, NEQ = neq, NIN = nin,     &
                            CX = CX, Y = Y, ITERS  = iters, MAXIT = maxit, &
                            GRADTOL = gradtol, FEASTOL = feastol,       &
                            PRINT_LEVEL = print_level )
!
! act on return status
!
      IF ( exit_code == 0 ) THEN                !  Successful return
         WRITE( 6, "( 1X, I0, ' iterations. Optimal objective value =',    &
     &    ES12.4, /, ' Optimal solution = ', ( 5ES12.4 ) )" ) iters, fx, X
      ELSE                                      !  Error returns
          WRITE( 6, "( ' LANCELOT_simple exit status = ', I0 ) " ) exit_code
      END IF
!
! clean up
!
      DEALLOCATE( X, BL, BU, CX, Y )
      DEALLOCATE( VNAMES, CNAMES )
!
      STOP
!
      END PROGRAM RUN_LANCELOT_simple
!
!...................................................................
!
      SUBROUTINE FUN ( X, F, i )
      INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
      REAL( KIND = wp ), INTENT( IN )   :: X( : )
      REAL( KIND = wp ), INTENT( OUT )  :: F
```

```
       INTEGER, INTENT( IN ), OPTIONAL   :: i
       IF ( .NOT. PRESENT( i ) ) THEN
!         the objective function value (user defined)
!=============================================================================
          F = 100.0_wp*(X(2)-X(1)**2)**2 +(1.0_wp-X(1))**2                    !
!=============================================================================
       ELSE
          SELECT CASE ( i )
          CASE ( 1 )
!             the equality constraint value (user defined)
!=============================================================================
              F = X(1)+3.0_wp*X(2)-3.0_wp                                     !
!=============================================================================
          CASE ( 2 )
!             the inequality constraint value (user defined)
!=============================================================================
              F = X(1)**2+X(2)**2-4.0_wp                                      !
!=============================================================================
          END SELECT
       END IF
       RETURN
       END SUBROUTINE FUN
!
!.....................................................................
!
       SUBROUTINE GRAD( X, G, i )
       INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
       REAL( KIND = wp ), INTENT( IN )  :: X( : )
       REAL( KIND = wp ), INTENT( OUT ) :: G( : )
       INTEGER, INTENT( IN ), OPTIONAL  :: i
       IF ( .NOT. PRESENT( i ) ) THEN
!          the objective function's gradient components (user defined)
!=============================================================================
           G( 1 ) = -400.0_wp*(X(2)-X(1)**2)*X(1)-2.0_wp*(1.0_wp-X(1))        !
           G( 2 ) =  200.0_wp*(X(2)-X(1)**2)                                  !
!=============================================================================
       ELSE
          SELECT CASE ( i )
          CASE ( 1 )
!             the equality constraint's gradient components (user defined)
!=============================================================================
              G( 1 ) =  1.0_wp                                                !
              G( 2 ) =  3.0_wp                                                !
!=============================================================================
          CASE ( 2 )
!            the inequality constraint's gradient components (user defined)
!=============================================================================
              G( 1 ) =  2.0_wp*X(1)                                           !
              G( 2 ) =  2.0_wp*X(2)                                           !
!=============================================================================
          END SELECT
       END IF
       RETURN
       END SUBROUTINE GRAD
!
!.....................................................................
!
       SUBROUTINE HESS( X, H, i )
       INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
       REAL( KIND = wp ), INTENT( IN )  :: X( : )
       REAL( KIND = wp ), INTENT( OUT ) :: H( : )
       INTEGER, INTENT( IN ), OPTIONAL  :: i
       IF ( .NOT. PRESENT( i ) ) THEN
```

```
!        the entries of the upper triangle of the objective function's
!        Hessian  matrix,  stored by columns  (user defined)
!=============================================================================
          H( 1 ) = -400.0_wp*(X(2)-3.0_wp*X(1)**2)+2.0_wp                    !
          H( 2 ) = -400.0_wp*X(1)                                            !
          H( 3 ) =  200.0_wp                                                 !
!=============================================================================
       ELSE
          SELECT CASE ( i )
          CASE ( 1 )
!             the entries of the upper triangle of the equality
!             constraint's Hessian matrix, stored by columns (user defined)
!=============================================================================
              H( 1 ) = 0.0_wp                                                !
              H( 2 ) = 0.0_wp                                                !
              H( 3 ) = 0.0_wp                                                !
!=============================================================================
          CASE ( 2 )
!            the entries of the upper triangle of the inequality
!            constraint's Hessian matrix, stored by columns (user defined)
!=============================================================================
              H( 1 ) = 2.0_wp                                                !
              H( 2 ) = 0.0_wp                                                !
              H( 3 ) = 2.0_wp                                                !
!=============================================================================
          END SELECT
       END IF
       RETURN
       END SUBROUTINE HESS
!.....................................................................
```

The use of the above calling program then produces the following output:

```
 8 iterations. Optimal objective value =  2.3314E-02
 Optimal solution =   8.4750E-01  7.1750E-01
```